CMPEN/EE 454, Project 2, Fall 2024

Due Friday Nov 15 11:59PM on Canvas

1 Motivation

This goal of this project is to help you understand in a practical way the course material on camera projection, triangulation, and epipolar geometry.

You will be given two views taken at the same time of a person performing a movement in a motion capture lab. The person has infrared-reflecting markers attached to his body, and there are several precisely synchronized and calibrated infrared cameras located around the room that triangulate the images of these markers to get accurate 3D point measurements. The two views you are given are from a pair of visible-light cameras that are also synchronized and calibrated with respect to the mocap system. As a result, we know the intrinsic and extrinsic camera calibration parameters of the cameras for each view, and these accurately describe how the 3D points measured by the mocap system should project into pixel coordinates of each of the two views.

2 Input Data

You are given the following things:

Two images, **im1corrected.jpg** and **im2corrected.jpg**, representing views taken at exactly the same time by two visible-light cameras in the mocap lab. These images have already been processed to remove nonlinear radial lens distortion, which is why they are called "corrected". Because the lens distortion has been removed, the simple, linear (when expressed in homogeneous coordinates) pinhole camera model we have studied in class gives a fairly accurate description of how 3D points in the scene are related to 2D image points and their viewing rays.

Two matlab files **Parameters_V1.mat** and **Parameters_V2.mat** representing the camera parameters of the two camera views (V1 and V2).

A matlab file **mocapPoints3D.mat** containing 3D point locations of 39 markers on the performer's body. These are measured with respect to a "world" coordinate system defined within the motion capture lab with the origin (0,0,0) located in the middle of the floor, positive Z-axis pointing vertically upwards, and units measured in millimeters.

3 Tasks to Perform

We want your group to perform the following 7 tasks using the images, mocap points and camera calibration data:

3.1 Understanding pinhole camera model parameters

Each of Parameters_V1.mat and Parameters_V2.mat contains a Matlab structure storing intrinsic and extrinsic calibration parameters of the pinhole camera model for each camera. Part of your job will be figuring out what the fields of the structure mean in regards to the pinhole camera model parameters we discussed in class lectures. Which are the internal parameters? Which are the external parameters? Which internal parameters combine to form the matrix Kmat? Which external parameters combine to form the matrix Pmat? Hint: the field "orientation" is a unit quaternion vector describing the camera orientation, which is also represented by the 3x3 matrix Rmat; you want to use Rmat for rotation in the pinhole camera matrix equations, not the quaternion. What is the location of the camera? Verify that the location of the camera and the rotation Rmat of the camera combine in the expected way (expected as per one of the slides in our class lectures on camera parameters) to yield the appropriate entries in Pmat.

3.2 Projecting 3D mocap points into 2D pixel locations

Write a function from scratch that takes the 3D mocap points and the camera parameters for an image and project the 3D points into 2D pixel coordinates for that image. You will want to refer to our lecture notes for the transformation chain that maps 3D world coordinates into 2D pixel coordinates. For verification, visualize your projected 2D points by plotting the x and y coordinates of your 2D points onto the image. If your projection function is working correctly, the points should be close to or overlapping the person's body, in many cases near the locations of visible markers attached to the person's body (other locations will be on markers that are not visible because they are on the side of the person that is facing away from the camera). If the plotted body points are grossly incorrect, such as outlining a shape much larger or smaller or forming a really weird shape that doesn't look like it conforms to the arms and legs of the person in the image), then something is likely wrong in your projection code. Illustrate that your projection code works correctly for both of the camera views.

3.3 Triangulation to recover 3D mocap points from two views

As a result of the step 3.2 you now have two sets of corresponding 2D pixel locations in the two camera views. Perform triangulation on each of pair of corresponding 2D points to estimate a recovered 3D point position. As per our class lecture on triangulation, this will be done for a corresponding pair of 2D points by using camera calibration information to convert each into a viewing ray represented by camera center and unit vector pointing along the ray passing through the 2D point in the image and out into the 3D scene. You will then compute the 3D point location that is closest to both sets of rays (because they might not exactly intersect). Go

back and refer to our lecture on triangulation to see how to do the computation. To verify that your triangulation code is correct, apply it to all of the 39 mocap points that you projected and compare how close your set of reconstructed 3D points come to the original set of 3D points you started with. You should get reconstructed point locations that are very close to the original locations. Compute a quantitative error measure such as mean squared error, which is the average squared distance between original and recovered 3D point locations. It should be very small.

3.4 Triangulation to make measurements about the scene

After you have verified that your triangulation code is correct, we can start using it to make additional 3D measurements in the scene. Specifically, clicking on matching points in the two views by hand, you can compute by triangulation the 3D scene location that projects to those points. Use this strategy to verify/answer the following geometric questions about the scene.

- Measure the 3D locations of at least 3 points on the floor and fit a 3D plane aX+bY+cZ+d = 0 to them. Verify that your computed floor plane is (roughly) the plane Z=0.
- Measure the 3D locations of at least 3 points on the wall that has white vertical stripes painted on it and fit a plane. What, approximately, is the equation of the wall plane?

Assuming the floor is Z=0, answer the following questions:

- How tall is the doorway?
- How tall is the person (or more precisely, how high is the top of their head at this moment)?
- There is a camera mounted on a tall tripod over near the striped wall; what is the 3D location of the center of that camera (roughly)?

3.5 Compute the Fundamental matrix from known camera calibration parameters

This task might be the hardest in a mathematical sense – compute the 3x3 fundamental matrix F between the two views using the camera calibration information given. To do this, you will need to determine from the camera calibration information what the location of camera2 is with respect to the coordinate system of camera1 (or vice versa) as well as the relative rotation between them, then combine those to compute the essential matrix E = R S, and finally pre and post multiply E by the appropriate film-to-pixel K matrices to turn E into a fundamental matrix F that works in pixel coordinates. You have all the camera information you need, but it is a little tricky to get E because although we know how the cameras are related to the same world coordinate system, we aren't directly told how the two camera coordinate systems are related relative to each other – some mathematical derivation is necessary to figure this out. For example, given the rotation matrices R1 and R2 of the two cameras, the row and columns of each of them is telling us how to relate camera coord axes to world coords and vice versa. How, then, would you combine them and/or their inverses/transposes to represent camera 2 axes with respect to the camera 1 coord system? Similarly, how do you compute the position of camera 2 with respect to the coordinate system of camera 1?

As a sanity check to see if a candidate solution for the F matrix is on the right track, use it to map some points in image 1 into epipolar lines in image 2 to see if they look correct. Also check the mapping of image 2 points into image 1 epipolar lines. You are welcome to adapt the section of code in the eight point algorithm demo (see task 3.6) that draws epipolar lines overlaid on top of images to do this visualization – you don't have to figure out how to plot epipolar lines from scratch.

3.6 Compute the Fundamental matrix using the eight-point algorithm

Use the eight point algorithm that was demo'ed in class and that is available in the matlab sample code section of our course website to compute a fundamental matrix by selecting matching points in the two views by hand. Recall that for best results you would like to choose points that spread out across the 3D scene as much as possible (and that it would be terrible idea to choose all the points on only a single plane, such as the floor). The output of the demo code will be a fundamental matrix, the points that were clicked in image 1, and the points from image 2. Show us the values printed for the F matrix and also the plots of epipolar lines it displays for both of the camera views. Does this epipolar geometry look reasonable?

Secondly, make a modified version of the eightpoint code that doesn't do Hartley preconditioning. As discussed in lecture, Hartley was the first to realize that, as a numerical algorithm, you have to pay attention to numerical conditioning of the data points you feed into the eightpoint algorithm, otherwise small errors in input can magnify into large errors in estimated F matrix. You will have to make changes to two places in the code, one to remove the effects of Hartley's normalization procedure, and one further down in the code to remove the "unnormalization" of F. Run this modified code with the same input images and sample points that you had clicked in the step above [recall that eightpoint.m returns the points that you clicked, so you can pass in exactly the same points you clicked previously]. Show us the values printed for the F matrix and also the plots of epipolar lines it displays for both of the camera views. Does this epipolar geometry look reasonable? Discuss.

3.7 Quantitative evaluation of your estimated F matrices

Just looking at drawings of the epipolar lines gives us an idea whether an F matrix is roughly correct, but how to quantitatively measure the accuracy? The *symmetric epipolar distance* (SED) is an error measure that evaluates in image coordinates how accurate an estimated fundamental matrix is based on mean squared geometric distance of points to corresponding epipolar lines. "Being immediately physically intuitive, this is the most widely used error criterion in practice during the outlier removal phase (OpenCV: Open Computer Vision Library, 2009; Snavely et al., 2008; VxL, 2009), during iterative refinement (Faugeras et al., 2001; Forsyth and Ponce, 2002; Snavely et al., 2008), and in comparative studies to compare the accuracy of different solutions (Armangu´e and Salvi, 2003; Forsyth and Ponce, 2002; Hartley and Zisserman, 2004; Torr and Murray, 1997). Besides being physically intuitive, SED has the merit of being efficient to compute." [from Fathy et.al., "Fundamental Matrix Estimation: A Study of Error Criteria"]. To compute SED, recall that we have a set of 39 accurate 2D point

matches generated in Task 3.2. Let the coordinates of one pair of those points be (x1,y1) in image 1 and (x2,y2) in image 2. For a given fundamental matrix F, compute an epipolar line in image 2 from (x1,y1) and compute the squared geometric distance of point (x2,y2) from that line.[1] Repeat by mapping point (x2,y2) in image 2 into an epipolar line in image 1 and measuring squared distance of (x1,y1) to that line. Accumulate these squared distances over all 39 known point matches and at the end compute the mean over all of these squared distances. That is the SED error to report for the F matrix.

Report the SED error for the three F matrices you have computed (you computed one F matrix in Task 3.5 and two different F matrices in Task 3.6). Verify that the error for the F matrix computed from known camera calibration information is much smaller than the error of either of the F matrices computed using the eight point algorithm. That is to be expected. Also report whether removing Hartley preconditioning changes the SED error, and by how much. Is using Hartley's numeric conditioning method a big win or did it not really have much of an effect?

By the way, as a practical use, if you were using an estimated F matrix to guide the search for point matches in two views, the sqrt of the SED error gives an idea of how far away from an epipolar line, in pixels, to expect to find a matching point. Thus, this value forms the basis for coming up with a distance threshold to use for rejecting "outlier" point matches based on the epipolar constraint.

Extra Credit 1 (5 pts) : Modify F matrix for cropped views

Going back to the two camera images, crop them to get rid of a lot of the empty lab space in the images, focusing attention more tightly around the person in the two views. Remember the parameters of the cropping rectangles used (for example, upper left corner and height/width of each rectangle), and figure out how to modify the camera intrinsic parameter K matrices to describe 3D to 2D projection into the pixel coordinates of these cropped images. Also compute an updated F matrix to map points to lines in these cropped images. Demonstrate that 3D to 2D projection works correctly in your cropped views using your modified camera parameters, and that your modified F matrix correctly depicts the epipolar geometry between the two cropped views.

Extra Credit 2 (5 pts): Generate a top-down view of the floor plane

Modify our sample code "planewarpdemo" in the matlab sample code section of our course to generate a higher resolution output than it currently does, for example, by setting the output (destination) image be comparable in number of rows/cols to the input (source) image. Also, note that one deficiency of this code is that the user has to "guess" what the shape of the chosen rectangle is when specifying the output. Write a new version that does not rely on user input, and that generates a top-down view of the floor plane that is accurate up to a similarity transformation (rotation, translation and isotropic scale) with respect to the 2D X-Y world

[1] Note: if (a,b,c) are the coefficients of a line and (x,y) is a point then the squared geometric distance of the point to the line is calculated as $(ax+by+c)^2 / (a^2+b^2)$.

coordinate system in the floor plane Z=0. Hint: how can you relate ground plane X-Y coordinates to 2D image coordinates in the source and in the destination images, given the known camera parameters of one or both views? Explain how you are generating your top-down view. Also, with regard to the resulting output image, what things look accurate and what things look weird? Could this kind of view be useful for analyzing anything about the performance of a person as they move around in the room?

4 What Code Can I Use?

The intent is that you will implement these tasks using general Matlab processing functions (https://www.mathworks.com/help/matlab/functionlist.html). You can also use and adapt code from our eight point algorithm and plane warp demo functions available on our Canvas web site. You MAY NOT use anything from the computer vision toolbox, or any third-party libraries/packages.

5 What to Hand In?

You will be submitting a single big zip file that contains your code and an illustrated written report with narrative explanation and analysis. Name your zip file Team###.zip where ### corresponds to your group's number in the project groups sign up sheet. ONLY ONE MEMBER FROM EACH TEAM needs to submit! We don't need multiple copies of the same thing.

CODE (50%)

- Please organize your code into separate scripts/functions that address each of the tasks, with names that make clear which does what, for example task3_1.m, task3_2.m, task3_3.m and so on, so we can easily find what code implements which task. If you have some other helper functions, please give them descriptive names.
- Include lots of comments in your functions so that we have a clear understanding of what it is doing and how it is doing it.
- Each task script/function should act like a little "demo" in that it produces an output that convincingly displays that it is coming up with a solution to the given task, producing not just a text or array output but, whenever possible, a visual display depicting the output results (for example, by showing images with points and epipolar lines superimposed on them). You should also show these visual results in the relevant sections of your written report.

WRITTEN REPORT in PDF format (50%)

- At top of first page, start by writing all team member names.
- Write a separate section for each of the 7 tasks, in order, with heading names like "Results for Task 3.1", and so on. For each task, write in that section what you did to

address the assigned task. Answer any questions that were asked in the task description; explain how you derived any equations; show your results and images generated; explain what code you had to write or how you modified our existing demo code, explain any implementation decisions you made that were clever or unusual (in your opinion). Give some thought into what you are showing as visual output and how you are explaining it in the text – you are trying to convince a reader that you have come up with a valid solution to the given task. If you weren't able to get a working solution to one of the tasks, this is chance to explain where the difficulty was.

- If you did either or both of the two extra credit tasks, also include a separate section for each those, labeled "Results for Extra Credit 1" and/or "Results for Extra Credit 2".
- Document what each team member did to contribute to the project. It is OK if you divide up the labor into different tasks (it is expected), and it is OK if not everyone contributes precisely equal amounts of time/effort on each project. However, if two people did all the work because the third team member could not be contacted until late in the development process, this is where you get to tell us about it, so the grading can reflect the inequity.